

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-101777

(43)Date of publication of application : 16.04.1996

(51)Int.Cl.

G06F 9/45

G06F 1/32

(21)Application number : 06-261394

(71)Applicant : TOSHIBA CORP

(22)Date of filing : 30.09.1994

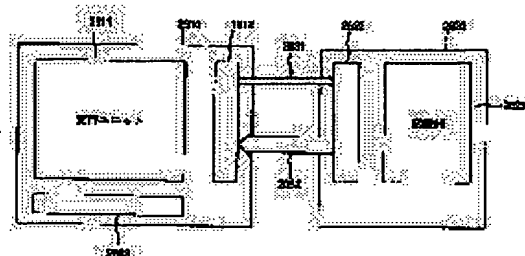
(72)Inventor : IDE YUKIHIRO
YOSHIDA TAKASHI

(54) INSTRUCTION STRING OPTIMIZING DEVICE

(57)Abstract:

PURPOSE: To perform the optimization processing for reduction of the power consumption in the stage of generation of a control program for information processor by providing an instruction string analysis means and an instruction change means.

CONSTITUTION: An intermediate code is generated, and next, it is discriminated whether each instruction has substitutive instructions registered in a library or not. When it is discriminated that the instruction has them, the library is retrieved to select the instructions which can be substituted for the original instruction. The power consumption of the original instruction and those of instructions retrieved from the library are tentatively calculated, and calculation results are compared with one another to select the instruction which minimizes the power consumption. It is discriminated whether optimization of all data is completed or not after the end of instruction substitution; and if it is completed, a program after the optimization processing is outputted. This control program is stored in a program memory 2620 of an information processor, and this program is used to control a CPU 2610, thus reducing the power consumption of an instruction bus 2632.



LEGAL STATUS

[Date of request for examination]

08.02.2000

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

3392545

[Date of registration]

24.01.2003

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

書誌

(19)【発行国】日本国特許庁(JP)
(12)【公報種別】公開特許公報(A)
(11)【公開番号】特開平8-101777
(43)【公開日】平成8年(1996)4月16日
(54)【発明の名称】命令列最適化装置
(51)【国際特許分類第6版】

G06F 9/45
1/32

【FI】

G06F 9/44 322 F 7737-5B
1/00 332 Z

【審査請求】未請求

【請求項の数】9

【出願形態】FD

【全頁数】26

(21)【出願番号】特願平6-261394

(22)【出願日】平成6年(1994)9月30日

(71)【出願人】

【識別番号】000003078

【氏名又は名称】株式会社東芝

【住所又は居所】神奈川県川崎市幸区堀川町72番地

(72)【発明者】

【氏名】井出進博

【住所又は居所】神奈川県川崎市幸区小向東芝町1 株式会社東芝研究開発センター内

(72)【発明者】

【氏名】吉田尊

【住所又は居所】神奈川県川崎市幸区小向東芝町1 株式会社東芝研究開発センター内

(74)【代理人】

【弁理士】

【氏名又は名称】佐藤 一雄(外3名)

要約

(57)【要約】

【目的】情報処理装置用制御プログラムの作成段階で消費電力を低減させるための最適化処理を行うことができる、命令列最適化装置を提供する。

【構成】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令について、相互の依存関係を解析する命令列解析手段と、この命令列解析手段で解析された依存関係に影響を与えない範囲で前記命令の順序を変更することによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段とを備える。

請求の範囲

【特許請求の範囲】

【請求項1】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令について、相互の依存関係を解析する命令列解析手段と、この命令列解析手段で解析された依存関係に影響を与えない範囲で前記命令の順序を変更することによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする命令列最適化装置。

【請求項2】前記プログラムを基本ブロックに分割して分割後の基本ブロックを前記命令列解析手段に送るブロック分割手段をさらに備えたことを特徴とする請求項1記載の命令列最適化装置。

【請求項3】前記ブロック内命令列変更手段が、直前に命令順序決定処理を行った基本ブロックの最後の前記ビット列と今回命令順序決定処理を行う基本ブロックの最初の前記ビット列との間のハミング距離を考慮して、このブロック内での命令順序決定処理を行うことを特徴とする請求項2記載の命令列最適化装置。

【請求項4】前記命令が、前記演算処理部によって前記プログラムが実行される際に考慮されない前記ビット列を含む場合に、このビット列に前後するビット列とのハミング距離が低減されるように、このビット列の信号値を変更することを特徴とする請求項1～3のいずれかに記載の命令列最適化装置。

【請求項5】データを一時的に記憶する複数のレジスタと、プログラムを記憶するプログラムメモリと、このプログラムメモリから命令バスを介して取り込んだ命令にしたがって前記レジスタに対するデータの書き込み／読み出しを行う演算処理部とを備えた情報処理装置が使用するためのプログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令中のレジスタ番号を認識するレジスタ番号認識手段と、このレジスタ番号認識手段で認識された前記レジスタ番号の有効範囲を認識するレジスタ有効範囲認識手段と、このレジスタ有効範囲認識手段が認識した前記有効範囲に影響を与えない範囲で前記レジスタ番号を変更することによって、このレジスタ番号を含む命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現

れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする命令列最適化装置。

【請求項6】前記命令列変更手段が、前記レジスタ番号認識手段が認識した各レジスタ番号について、前記レジスタ有効範囲認識手段が認識した前記有効範囲に影響を与えることなく置き換えることができるレジスタ番号を策定する策定手段と、前記レジスタ番号認識手段が認識したレジスタ番号および前記策定手段が策定したレジスタ番号のうち、前記命令バスに現れるビット列間のハミング距離が最も小さくなるレジスタ番号を選択する選択手段と、前記プログラム中のレジスタ番号を前記選択手段が選択したレジスタ番号と置き換える置換手段と、を備えたことを特徴とする請求項5記載の命令列最適化装置。

【請求項7】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令の一部または全部について、同じ命令を意味する他のビットパターンを記憶する記憶手段と、前記プログラム中の命令を前記記憶手段に記憶されたビットパターンに置換えることによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする命令列最適化装置。

【請求項8】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、前記プログラム中の命令または命令列を前記選定手段で選定された命令または命令列と置換えることによって、この命令または命令列を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする命令列最適化装置。

【請求項9】プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、前記プログラム中の命令または命令列および前記選定手段で選定された命令または命令列について、これらの命令または命令列を前記プログラムメモリから前記演算処理部に転送する際の前記命令バスにおける消費電力を試算する演算手段と、前記選定手段で選定された命令または命令列のうち、前記演算手段で試算された消費電力が前記プログラム中の命令または命令列よりも小さいものを、このプログラム中の命令または命令列と置換える命令列変更手段と、を備えたことを特徴とする命令列最適化装置。

詳細な説明

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、例えば情報処理装置の制御プログラム等を最適化するための命令列最適化装置に関するものである。

【0002】

【従来の技術】近年のマルチメディアの発展に伴って、例えばブックコンピュータ、ノートコンピュータ、携帯電話等の携帯型情報処理装置が大きな普及を見せている。

【0003】このような携帯型情報処理装置の制御部の概略構成を、図26に示す。同図に示したCPU(Central Processing Unit) 2610において、実行ユニット2611は、制御プログラムを構成する各命令を実行する。また、入出力部2612は、実行ユニット2611が実行する命令のアドレスを順次アドレスバス2631に出力するとともに、このアドレスに対応する命令バス2632から取り込む。レジスタ部2613は、実行ユニット2611による命令の実行に伴って生じたデータを一時的に記憶する。

【0004】一方、プログラムメモリ2620において、記憶部2621には、制御プログラムを構成する各命令が予め記憶されている。また、入出力部2622は、アドレスバスから入力されたアドレスに対応する命令を記憶部2621から読み出して、命令バス2632に出力する。

【0005】このような構成において、CPU2610による制御を行う際には、まず、実行ユニット2611が実行する命令のアドレスが、アドレスバス2631を介して、CPU2610の入出力部2612からプログラムメモリ2620の入出力部2622に送られる。これにより、プログラムメモリ2620は、指定されたアドレスに対応する命令を記憶部2621から読み出し、入出力部2622から出力する。CPU2610の入出力部2612は、この命令を命令バス2632を介して入力し、実行ユニット2611に送る。そして、実行ユニット2611がこの命令を実行することによって、情報処理装置の制御を行っている。

【0006】このようにして、実行ユニット2611が命令の実行を行っているときに、一時的に記憶する必要のあるデータが生じた場合、このデータはCPU2610内のレジスタ部2613に格納されるとともに、必要に応じて読み出される。

【0007】ここで、図26に示したような情報処理装置に使用する制御プログラムは、予め作成されて、情報処理装置の製造時に、プログラムメモリ2620に格納される。かかる制御プログラムの開発においては、高級言語或いはアセンブリ言語のコンパイル等を行って目的プログラム(すなわちプログラムメモリ2620に格納するプログラム)を作成する手段において、様々な最適化処理が施される。この最適化処理としては、例えば、制御プログラムの実行時間を短縮するための最適化処理や、この制御プログラムの格納に使用されるメモリ領域を低減させるための最適化処理などが、すでに知られている。

【0008】

【発明が解決しようとする課題】上述したような情報処理装置においては、従来より、消費電力の低減が要求されている。特に、携帯型の情報処理装置では、連続使用できる時間を向上させるために、消費電力の低減が要求されている。また、携帯型以外の情報処理装置においても、環境保全やエネルギー消費削減の観点から、消費電力の低減が要求されている。

【0009】情報処理装置の内部回路の消費電力 P は、以下のような式で表される。

【0010】 $P = \alpha \cdot C \cdot V_{dd}^2 \cdot n \cdot f + P_s$ ここで、 α は稼働率、 C は回路全体のキャパシタンス、 V_{dd} は電源電圧、 n は回路の素子数、 f は動作周波数、 P_s は待機時の消費電力である。

【0011】従来は、これらの各パラメータの値が小さくなるようにハードウェアを構成することによって、消費電力の低減が図られていた。

【0012】しかしながら、このようなハードウェア上の措置だけでは、消費電力を十分に低減させることはできなかった。

【0013】本発明は、このような従来技術の欠点に鑑みてなされたものであり、情報処理装置用制御プログラムの作成段階で消費電力を低減させるための最適化処理を行うことができる、命令列最適化装置を提供することを目的とする。

【0014】

【課題を解決するための手段】

(1) 第1の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令について、相互の依存関係を解析する命令列解析手段と、この命令列解析手段で解析された依存関係に影響を与えない範囲で前記命令の順序を変更することによって、この命令を前記プログラムメモリから前

記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

(2) 第2の発明に係わる命令列最適化装置は、データを一時的に記憶する複数のレジスタと、プログラムを記憶するプログラムメモリと、このプログラムメモリから命令バスを介して取り込んだ命令にしたがって前記レジスタに対するデータの書き込み／読み出しを行う演算処理部とを備えた情報処理装置が使用するためのプログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令中のレジスタ番号を認識するレジスタ番号認識手段と、このレジスタ番号認識手段で認識された前記レジスタ番号の有効範囲を認識するレジスタ有効範囲認識手段と、このレジスタ有効範囲認識手段が認識した前記有効範囲に影響を与えない範囲で前記レジスタ番号を変更することによって、このレジスタ番号を含む命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

(3) 第3の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラムを構成する各命令の一部または全部について、同じ命令を意味する他のビットパターンを記憶する記憶手段と、前記プログラム中の命令を前記記憶手段に記憶されたビットパターンに置換えることによって、この命令を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

(4) 第4の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、前記プログラム中の命令または命令列を前記選定手段で選定された命令または命令列と置換えることによって、この命令または命令列を前記プログラムメモリから前記演算処理部に転送する際に前記命令バスに現れるビット列間のハミング距離を低減させる命令列変更手段と、を備えたことを特徴とする。

(5) 第5の発明に係わる命令列最適化装置は、プログラムを記憶するプログラムメモリとこのプログラムメモリから命令バスを介して前記プログラムを取り込む演算処理部とを備えた情報処理装置が使用するための前記プログラムを最適化する命令列最適化装置において、前記プログラム中の命令または命令列について、同じ処理結果を得ることができる他の命令または命令列を選定する選定手段と、前記プログラム中の命令または命令列および前記選定手段で選定された命令または命令列について、これらの命令または命令列を前記プログラムメモリから前記演算処理部に転送する際の前記命令バスにおける消費電力を、ハミング距離を考慮して試算する演算手段と、前記プログラム中の命令または命令列を前記選定手段で選定された命令または命令列と置換えることによって、前記演算手段が試算した消費電力を低減させる命令列変更手段と、を備えたことを特徴とする。

[0015]

[作用]

(1) 第1の発明に係わる命令列最適化装置によれば、プログラムを構成する各命令について相互の依存関係を解析し、この依存関係に影響を与えない範囲で命令の順序を変更することによってこの命令をプログラムメモリから演算処理部に転送する際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

(2) 第2の発明に係わる命令列最適化装置によれば、プログラムを構成する各命令のレジスタ番号を認識し、続いて、このレジスタ番号の有効範囲を認識し、そして、この有効範囲に影響を与えない範囲でレジスタ番号を変更することによって、このレジスタ番号を含む命令が転送される際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

(3) 第3の発明に係わる命令列最適化装置によれば、同じ命令を意味する他のビットパターンをプログラム中の命令と置換えることによって、この命令をプログラムメモリから演算処理部に転送する際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

(4) 第4の発明に係わる命令列最適化装置によれば、同じ処理結果を得ることができる他の命令または命令列をプログラム中の命令または命令列と置換えることによって、この命令または命令列をプログラムメモリから演算処理部に転送する際に命令バスに現れるビット列間のハミング距離を低減させることとしたので、情報処理装置の消費電力を低減させることができる。

(5) 第5の発明に係わる命令列最適化装置によれば、プログラム中の命令または命令列、および、この命令または命令列と同じ処理結果を得ることができる他の命令または命令列の消費電力を試算し、この消費電力が小さい命令または命令列に置換えることとしたので、情報処理装置の消費電力を低減させることができる。

[0016]

[実施例] 以下、本発明の実施例について、図面を用いて説明する。

[0017] (実施例1) 実施例1として、第1の発明の一実施例(請求項1~4に対応する)について説明する。

[0018] 図1は、第1の発明(請求項1に対応する)の概念を概略的に示すフローチャートである。同図に示したように、第1の発明においては、まず、プログラムを構成する各命令について、命令列解析手段を用いて、相互の依存関係を解析する(ステップS100)。そして、この依存関係に影響を与えずに命令バスに現れるビット列間のハミング距離が低減されるように、命令の順序を変更する(ステップS101)。そして、この命令順序の変更により、命令バスにおける消費電力の低減が実現される。

[0019] このステップS101では、まず、消費電力低減化手段(第1の発明の「命令列変更手段」に相当する)を用いて、命令順序の変更と、このときのハミング距離の判定とを行う(ステップS102)。次に、判定されたハミング距離を所定の基準値と比較する(ステップS103)。ここで、基準値は、予め定められた値でもよいし、それまでに消費電力低減化手段で判定されたハミング距離の最低値であってもよい。そして、消費電力低減化手段で判定されたハミング距離が基準値よりも小さい場合は、この命令の順序を最適化の結果として出力し、この最適化処理を終了する(ステップS104)。一方、判定されたハミング距離が基準値よりも大きい場合は、他の命令順序について、同様の処理(ステップS102、S103)を繰り返す。

[0020] 次に、図1を具体化した例(請求項1~3に対応する)について、図2のフローチャートを用いて説明する。

[0021] なお、ここでは、実行ユニットおよび命令バスがともに32ビットの情報処理装置(図14(a)参照)で用いられる制御プログラムの最適化に第1の発明を適用した場合を例にとって説明する。

[0022] 図2において、ステップS200では、制御プログラムを、基本ブロックに分割する。ここで、基本ブロックとは、例えば式や代入文の並びのような、途中から外部への分岐が起こらず、また、外部から途中への分岐も起こらないプログラムブロックをいう。基本ブロックへの分割が終了すると、次に、この基本ブロック内の各命令シーケンスに対して、それらの命令の依存関係およびレジスタの依存関係を解析し、命令の入れ換えによって処理の因果律を侵さない範囲を特定する。そして、このようにして特定された各範囲を識別するための識別子をプログラムに付加する。

[0023] ステップ200における処理は、従来から知られているような他の最適化処理においても利用される処理であるので、ここでは詳細の説明を省略する。このような処理を開示した文献としては、例えば以下のようなものがある。

[0024] Z.Li and P.-C.Yew, "Efficient Interprocedural Analysis for Program Parallelization and Restructuring," Proc.ACM SIGPLAN PPEALS, pp.85-97, 1988. S.Jain and C.Thompson, "An Efficient Approach to Dataflow Analysis in a Multiple Pass Global Optimizer," Proc.SIGPLAN'88 Conf.on Prog.Lang.Design and Implementation(PLDI'88), pp.154-163, 1988. 次に、ステップS201で、基本ブロック

ごとに最適化処理を行う。以下、このステップS101の処理手順について説明する。

【0025】まず、ステップS202により、初期設定を行う。図2において、変数LastComは、前回に最適化処理を行った基本ブロックの最後の命令が代入されている。初期設定においては、この変数LastComに、デフォルト値(命令ビット列のひとつ)を代入する。このデフォルト値としては、どのような値を使用してもよく、例えば、全ビット“1”や全ビット“0”であってもよい。但し、現実的には、統計的に最も頻繁に出現する命令や、最終的にリンクされるヘッダ・プログラム(プロローグ・プログラム、すなわち、OSから目的のユーザが作成したプログラムを起動させ、実行終了後のOSに戻るためのプログラム)或いはランタイム・ルーチンなどの最終命令などをデフォルト値とすることが好ましい。

【0026】以降のステップS203～S212では、ハミング距離を最小にするための処理が行われる。ここで示す手順は、命令の順番を順次変更し、可能な全パターンを試行することによって最適化を行う、素朴且つ確実な「しらみつぶし」による方法を用いている。なお、複雑なデータ構造の基本ブロックの場合は、特別な手法を用いて高速処理を行うことも有効であるが、本発明では特に限定されるものではない。

【0027】ステップS203においては、全ての処理が終了したか否かの判定を行う。本実施例における最適化処理は、基本ブロックごとに行われるので、すべての基本ブロックに対する処理が終了した時点で終了する。この判定の結果、処理を行っていない基本ブロックが残っている場合には、ステップS204以降の処理を実行する。

【0028】ステップS204では、基本ブロック内の最適化処理が終了したか否かの判定を行う。この最適化処理は、基本ブロック内の各命令の実行順序を置き換えることによって行われる。注目している基本ブロックに対する処理は、命令の順序依存やレジスタ依存などに矛盾を生じさせない全ての可能な置き換え方(順列)について試行し終わっている場合に終了し、ステップS213に進む。一方、全ても置き換え方の試行を終了していない場合は、ステップS205以降の最適化処理を続行する。

【0029】ステップS205では、ブロック内の最適化に先立つ初期設定を行う。図2において、変数Hd_sumは、注目している基本ブロック内での命令間のハミング距離の総和を示し、変数Hd_bounは変数LastComと注目している基本ブロックの先頭命令とのハミング距離である。また、変数Hd_totalは、変数Hd_sumと変数Hd_bounとの和を示し、変数Hd_minは、変数Hd_totalの最小値を示している。この変数Hd_minに、初期設定として、“∞”を代入する。ここで、“∞”はいかなる数字よりも大きい値であるものとする。

【0030】ステップS206では、注目している基本ブロック内のハミング距離の総和を求め、変数Hd_sumに代入する。この総和は、隣接する命令のビット・パターンを比較し、対応する桁のビットの値が異なっている場合を“1”として累積加算することにより簡単に求めることができる。

【0031】ステップS207では、ステップS206と同様にして、前回処理した基本ブロックの最後の命令(変数LastCom)と今回注目している基本ブロックの先頭命令とのハミング距離を求め、変数Hd_bounに代入する。この操作を行うことによって、基本ブロック間にまたがる最適化処理が可能となる。

【0032】ステップS208では、現在の命令シーケンス・パターンにおける命令間のハミング距離の総和、すなわち変数Hd_sumと変数Hd_bounとの和を算出して、変数Hd_totalに代入する。

【0033】ステップS209では、ステップS208で得られたHd_totalの値と、現在までの試行によって得られたHd_totalのうちの最小値であるHd_minとが、比較される。そして、 $Hd_total \geq Hd_min$ の場合は、異なる命令シーケンス・パターンについての試行をさらに行うべく、ステップS212以降を実行する。一方、 $Hd_total < Hd_min$ の場合は、ステップS210、S211を実行した後、ステップS212以降を実行する。

【0034】ステップS210、S211では、変数の更新を行う。まず、ステップS210においては、Hd_totalの値をHd_minに代入する。また、ステップS211においては、このHd_minに対応する命令シーケンス・パターンを、変数MinHdSequenceに記憶する。

【0035】ステップS212においては、命令シーケンスの入れ換えを行う。ここでは、基本ブロック内の命令の順番を入れ換えることにより、まだ試行していない命令シーケンスを生成して、ステップS204に戻る。この命令の入れ換えにおいては、ステップS200で行った解析の結果に基づいて、因果律に矛盾が生じないようにする。

【0036】以上説明したステップS203～S212からなる処理を反復して行うことにより、最適化を行うことができる。

【0037】すべての命令シーケンス・パターンについての試行を終了すると(ステップS204)、続いて、ステップS213を実行する。ステップS213では、最適化された命令シーケンス(変数MinHdSequenceに記憶されている)を、最適化の結果として出力する。

【0038】ステップS214では、変数LastComの更新を行う。すなわち、この変数LastComに、変数MinHdSequenceに記憶された最後の命令を代入する。

【0039】そして、以上の説明と同様にして、次の基本ブロックについて、ステップS203以降の処理を実行する。

【0040】次に、本実施例の最適化処理を実際に行う場合について、図3に示したようなプログラムを用いる場合を例にとって説明する。

【0041】図3のプログラムは、整数の内積演算を行うためのプログラムであり、C言語で記述されている。

【0042】図4～図7は、図3に示したプログラムを“Sun SPARC コンパイラ”でコンパイルした場合のアセンブリ・ソース・プログラムリストを示している。図4～図7において、第1カラムはライン番号、第2カラムはアドレス、第3カラムはオブジェクト・コード、第4カラムはアセンブリ・ソースを、それぞれ示している。なお、C101～C313およびB1～B6は、説明のための符号である。ここで、アセンブリ文法やニーモニック等については、公知技術であるので説明を省略するが、これらを開示した文献としては例えば以下のようなものがある。

【0043】SPARC International, Inc., The SPARC Architecture Manual Version 8, Prentice-Hall, Inc. A Simon and Schuster Company Englewood Cliffs, New Jersey 07632. 本実施例の最適化処理装置に入力されるのは、図4～図7に示したようなアセンブリ・ソースである。そして、このアセンブリ・ソースは、図2にステップS200によって基本ブロックが検索される。これにより、このプログラムは、図4～図7にB1～B6で示したような6個の基本ブロックに分割される。

【0044】さらに、各基本ブロック内の命令の依存関係が解析される。図8および図9は、依存関係の解析結果を示す有向グラフである。ここで、図8は基本ブロックB1(図4参照)の有向グラフであり、図4のC101～C104はそれぞれ図8のノードN101～N104に対応している。同様に、図9は基本ブロックB3(図5参照)の有向グラフであり、図5のC301～C313はそれぞれ図9のノードN301～N313に対応している。また、ノードN100、N300は有向グラフのトップを表すダミー・ノードであり、ノードN105、N314は有向グラフのボトムを表すダミー・ノードである。さらに、アークA101～A105、A301～A314は命令の依存関係を示し、矢印の逆の順番に命令を実行してはならないことを示している。基本ブロック内の命令の順序関係は、このような有向グラフを作成することによって管理する。

【0045】まず、第1の基本ブロックB1の最適化処理について説明する。基本ブロックB1は、本プログラムを実行するために内部状態を保存するためのプロローグ処理である。図8の有向グラフに示したように、4個の命令C101～C104は、この順番でしか実行できないので、そのまま出力される。

【0046】第2の基本ブロックB2の最適化処理については説明を省略するが、この基本ブロックB2の処理の終了時には、最後の命令C204としての“nop”がLastComに記憶されている。

【0047】次に、第3の基本ブロックB3の最適化処理を行う。基本ブロックB3においては、図9の有向グラフに示したように、“C301”、“C302～C304”、“C305”、“C306～C308”相互間で命令シーケンスの順序を入れ換えることは可能であるが、“C302

～C304”や“C306～C308”内で命令の順序を入れ換えることはできない。また、C309～C313は、この順番でしか実行できず、C301～C308よりも先に実行することはできない。

【0048】まず、命令シーケンス・パターンを図5のとおり、すなわちC301、C302・・・C313とした場合について、図2に示したような最適化処理ステップS205～S211を行う。このとき、LastCom1には、上述したように、基本ブロックB2の最後の命令C204としての“nop”が記憶されている。また、最小ハミング距離Hd_minには、初期設定として“∞”が代入されている。

【0049】図10は、基本ブロックB2の最後の命令C204および基本ブロックB3の各命令C301～C313のビット・パターンを示している。このようなビット・パターンについて、Hd_sum、Hd_boun、Hd_totalを算出すると(図2のステップS206～S208参照)、Hd_sum=161、Hd_boun=13となり、したがってHd_total=174となる。

【0050】ここで、最小ハミング距離Hd_min=∞であるので、ステップS209(図2参照)での比較の結果、Hd_minにはHd_total=174が代入され(ステップS210)、さらに、図5に示したような命令シーケンスが変数MinHdSequence1に記憶される(ステップS211)。

【0051】次に、命令の順序を入れ換えた場合について、同様の処理(ステップS205～S211)を行う。

【0052】そして、入れ換えが可能な命令シーケンスのすべてについての試行が終了すると、変数MinHdSequence1に記憶されている命令シーケンスを、命令シーケンスの最適化の結果として出力する(ステップS213)。

【0053】ステップS213で出力された命令シーケンスのビットパターンを図11に示す。また、参考例として、最悪の(すなわち、ハミング距離の総和が最大になる)ビットパターンを図12に示す。図11におけるハミング距離の総和Hd_totalは130となる。また、図12におけるハミング距離の総和Hd_totalは196となる。すなわち、本実施例によれば、最適化処理によって、基本ブロックB3を実行する際の命令バスのスイッチング回数を最適化前の74.7%とすることができ、また、最悪の場合の66.3%とすることができた。

【0054】以下、同様にして第4～第6の基本ブロックB4～B6についての最適化処理を行うが、これらの各ブロックB4～B6内では命令の順番を入れ換えることはできないので、そのまま出力して処理を終了する。

【0055】次に、本実施例に係わる命令列最適化装置の変形例(請求項4に対応する)について、図13を用いて説明する。

【0056】プログラム中の命令によっては、命令フォーマットの中に“do not care”のビット、すなわち“1”あるいは“0”のどちらであっても、その命令の動作に影響を与えないビットを含む場合がある。例えば、上述の基本ブロックB3において、命令C303、C304、C307、C308の12ビット目から6ビット目(ビット(11:5))は、“do not care”のビットである(図10参照)。このようなビットの値を適当に変更することによって、隣接する命令間でのハミング距離を低減させることができる場合がある。

【0057】図13は、“do not care”のビットの値を変更することによってハミング距離を低減させるための処理の一例を示すフローチャートである。

【0058】図13に示したような処理を、図2のステップS206に換えて実行することにより、“do not care”のビットを考慮して、さらなるハミング距離の低減を図ることができる。

【0059】同図において、ステップS1301では、初期設定として、変数Hd_sumに初期値“∞”を代入する。

【0060】次に、ステップS1302において、本処理が終了したか否かの判定を行う。本処理においては、“do not care”の全ビットの値を変更しつつ、以下のような試行を行う。そして、“do not care”のビットの“1”、“0”の組み合わせについてのすべての試行を終了すると、本処理を終了する。

【0061】ステップS1303では、現時点での“do not care”のビット・パターンについて、隣接する命令間でのハミング距離の総和を求め、変数Hd_sum_currentに代入する。

【0062】ステップS1304では、Hd_sumとHd_sum_currentとの大小比較を行う。ここで、 $Hd_sum \geq Hd_sum_current$ であれば、ステップS1305でHd_sumにHd_sum_currentの値を代入したのち、ステップS1306へ進む。一方、 $Hd_sum < Hd_sum_current$ であれば、ステップS1305を実行することなく、そのままステップS1306へ進む。

【0063】ステップS1306では、“do not care”のビット・パターンを、まだ試行していないビット・パターンに変更する。

【0064】このような処理を、例えば基本ブロックB3の最適化前のプログラム(図10参照)においては、ハミング距離の総和Hd_sum、Hd_totalを10だけ低減させることができる。

【0065】以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてCPU等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【0066】本実施例に示した手順は、最適化を入力データに対して順次実行する1パス方式のものである。そのため、最終的に得られた命令列は、ハミング距離が最小でない場合もあり得る。これは、基本ブロックの境界の最適化において、注目している基本ブロックの一つ前の基本ブロックの最終命令しか考慮していないためである。したがって、ハミング距離をさらに低減させるためには、例えば、注目している基本ブロックの次の基本ブロックやさらに次の基本ブロック等をも考慮する方法等が考えられる。しかし、単純な処理で迅速に最適化を行うためには、基本ブロックに分割して処理を行う方が望ましい。

【0067】また、アセンブリ・ソースに対して本実施例に係わる最適化処理と他の最適化処理(例えば、制御プログラムの実行時間を短縮するための最適化処理や、この制御プログラムの格納に使用されるメモリ領域を低減させるための最適化処理など)を行う場合には、各最適化処理を行う順番に係わらず、本実施例の効果を得ることができる。しかし、本実施例の効果を最も有効に得るためには、本実施例に係わる最適化を最後に行うことが望ましい。ここで、本実施例の最適化処理を最後に行う場合には、他の最適化処理の結果を変更してしまう場合が考えられるが、このような不都合の防止は、ステップS100(図1参照)の依存解析のフェイズにおいて考慮すればよい。すなわち、図8、図9に示したような有向グラフにおいて、他の最適化処理の結果を変更しないように制約の設定を行えばよい。

【0068】以上説明した本実施例では、実行ユニットおよび命令バスがともに32ビットの情報処理装置で使用する制御プログラムの最適化を例にとった。すなわち、本実施例で最適化された制御プログラムは、図14(a)に示したような、命令を一度に1個ずつ読み出して、フェッチ、発行、デコード、実行を行う情報処理装置の制御に使用されることを前提としていた。したがって、最適化処理において考慮すべきハミング距離は、図15(a)に示すように、隣接する命令間のハミング距離である。

【0069】しかしながら、図14(b)に示したように、今日ではCPUの多くは一度に複数の命令の読み出し、フェッチ、発行が行える構成となっている。このようなCPUを用いる場合、最適化を行う際に考慮すべきハミング距離は、隣接する命令間のハミング距離ではなく、命令バスの同じフィールド、同じビット位置に割り当てられる命令間のハミング距離である。すなわち、図14(b)に示したように、一度に2命令ずつ命令を読み出すような構成のCPUの制御プログラムを最適化する場合は、図15(b)に示したように、1つおきの命令間(例えば、C301とC303、C302とC304等)のハミング距離が低減されるように最適化を行えばよい。このような場合、2個ずつの命令をビット結合(コンカチネート)して得られたビット列を作成することとすれば、本実施例の最適化装置をそのまま用いて最適化処理を行うことができる。

【0070】図16(a)は、一度に4命令ずつ命令を読み出すような構成の装置を示している。このような場合も、図16(b)に示したように、3つおきの命令間(例えば、C301とC305、C302とC306等)のハミング距離が低減されるように最適化を行えばよい。そして、4個ずつの命令をビット結合して得られたビット列を作成することとすれば、本実施例の最適化装置をそのまま用いて最適化処理を行うことができる。

【0071】また、情報処理装置によっては、内部の命令バスのバンド幅と外部の命令バスのバンド幅とが異なる場合がある。図17

は、内部の命令バスのバンド幅は128ビットであるが、外部の命令バスのバンド幅は32ビットである場合を示している。このような場合には、内部バスについては3命令おきの命令間でハミング距離の低減を行い、外部バスについては隣接する命令間でハミング距離の低減を行えばよい。どちらを優先するか、或いは互いに妥協するのかは、消費電力の低減に対する関与の度合い等に応じて、適宜決定すればよい。

【0072】(実施例2)実施例2として、第2の発明の一実施例(請求項5、6に対応する)について説明する。

【0073】なお、ここでは、実行ユニットおよび命令バスがともに32ビットの場合を例にとりて説明する。

【0074】本実施例の命令列最適化装置においては、レジスタに割り当てられるべき変数に注目して、制御プログラムの最適化を行う。すなわち、その変数が現れる命令列のある区間でビットの変化量を見て、最小となるレジスタ番号を割り当てていく。

【0075】本実施例では、 $c=a-b$ 、 $c=a/b$ 等において、 a をソース、 b をターゲット、 c をデスティネーションとし、 a の値を保持するレジスタをソース・レジスタ、 b の値を保持するレジスタをターゲット・レジスタ、 c の値を保持するデスティネーション・レジスタと称することとする。そして、32ビットの命令のうち、左側MSBから数えて、1ビット目～10ビット目および21ビット目～27ビット目を命令コードのフィールドとし、11ビット目～15ビット目をデスティネーション・レジスタのフィールドとし、16ビット目～20ビット目をソース・レジスタのフィールドとし、27ビット目～32ビット目をターゲットレジスタのフィールドとする。

【0076】あるレジスタ番号に格納されるデータの有効範囲は、このレジスタ番号がデスティネーションのレジスタ番号として現れる命令によってデータがレジスタに格納されてから、この格納データが必要とされる命令、すなわち、このレジスタ番号がソースレジスタまたはターゲットレジスタとして現れる命令までである。当然ながら、一つのレジスタ番号は複数の変数或いはデータの一時記憶場所として使い回しをして効率化を図っている。したがって、プログラム中で、1個のレジスタ番号がある一つのレジスタデータを保管している範囲を知るには、コンパイラの最適化部でレジスタ番号を割り当てる際に作られるレジスタ割当テーブルを解析し、有効範囲テーブルを作製することが必要となる。

【0077】有効範囲テーブルは、レジスタ割当テーブルから容易に作製することができる。通常、コンパイラでは、ソースプログラムからデータフローグラフ或いは依存グラフを生成する。そして、このグラフを用いて、ある変数のデータまたは一時的な演算の中間結果を示すデータの保持のためにレジスタを割り当てていく。ここで、従来は、変数が現れるとレジスタを割り当てテーブルに登録し、データフローグラフ或いは依存グラフから不要になったと判断された変数についてはレジスタの割当テーブルからエントリを削除していた。そして、これにより、あるレジスタ番号のレジスタに保持されているデータが有効なアドレスの範囲、すなわち、あるロード命令或いは演算結果の書き込みにより有効なデータが書き込まれてからそのデータを必要とする最後の演算命令或いはストア命令が出現するまでの有効期間を判断していた。

【0078】レジスタ番号を割り当てる際に、例えば、 $c=a+bc=c*d$ というソースプログラムがあった場合、 a :レジスタ0 b :レジスタ1 c :レジスタ2 d :レジスタ3とし、コンパイル後のプログラムを(1)

add r0, r1, r2mul r2, r3, r2としてもよいが、乗算結果としての c のみをレジスタ5に格納すること、すなわち、(2)

add r0, r1, r2mul r2, r3, r5としてもよい。この場合には、ソースプログラムの変数 c に対し、複数のレジスタ番号を割り当てることになる。従来は、上記プログラム(1)のように、例えばSPARCのレジスタウィンドウのグローバルレジスタのようにレジスタに特殊な意味または機能がある場合を除いては、レジスタリソースの問題からレジスタからのデータの退避が必要ないときには複数のレジスタ番号への割り付けをできるだけ避けている。これに対して、本実施例では、上記プログラム(2)のように、複数のレジスタへの割り当てを行うものとする。これにより、評価対象の評価範囲を分割することができるので、評価範囲が狭まり、評価対象を増加させることができる。一方、上述のようにレジスタに特殊な意味・機能がある場合には、もともと割り当てられていたレジスタと同じ機能を有するレジスタ以外には割り当てることができないので、選択範囲が狭くなり、注意が必要である。

【0079】有効範囲テーブルを作成した後は、注目するレジスタ番号についてハミング距離を求め、割り付け可能な他のレジスタ番号の再割り当てを行う。この再割り当てを行う際には、さらに複数のレジスタに割り当ててを試みることで、評価範囲の分割を図ることも可能である。

【0080】また、本実施例では、複数のレジスタ番号について同時に評価することも可能である。同時に評価を行うレジスタ番号を1つに限定すると、置き換えが可能なレジスタ番号の数が限定されるが、複数のレジスタ番号について同時に評価することにより、これらの複数のレジスタ番号を互いに置き換えが可能なレジスタ番号として置き換えの最適化を行うことができる。

【0081】さらに、レジスタファイルの再割り当てを行う段階は、コンパイラでレジスタ割り当てを行うときでもよいし、一旦割り当てを行った後であってもよい。

【0082】次に、本実施例の命令列最適化装置の具体的な例について、図18～図20を用いて説明する。

【0083】図18は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【0084】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード(アセンブリ・コード)を作製する(ステップS1801)。このようにして得られた中間コードのプログラム例を、図19(a)に示す。

【0085】次に、レジスタ割当テーブルを作製し、さらに、このレジスタ割当テーブルから有効範囲テーブルを作製する(ステップS1802)。図19(a)に示したプログラムの有効範囲テーブルを、図20に示す。

【0086】そして、今回の試行で注目するデータを選択し(ステップS1803)、この注目データに割当られたレジスタ番号を、ハミング距離が最低となるようなレジスタ番号と置き換える(ステップS1804)。このとき、有効範囲の境界での命令のハミング距離も考慮し、この境界におけるハミング距離も小さくなるような最適化を行う。

【0087】ここで、レジスタ番号0x1cに割り当てられていたデータが注目データである場合について考える。このデータの有効範囲は、図20の有効範囲テーブルより、アドレス0101からアドレス1000までであることがわかる。すなわち、かかるデータについては、アドレス0101～1000について評価すればよい。なお、図19(a)のプログラムでは、アドレス1101の命令でレジスタ番号0x1cに新たなデータが格納されているが、評価対象外なので評価しない。

【0088】図19(a)のプログラムでは、レジスタ番号0x1cの前後でのハミング距離の総和は14となっている。ここで、ハミング距離の総和を最小にする他のレジスタ番号を探すと、レジスタ番号0、2に置き換えることによってハミング距離の総和を8にできることがわかる。ここでは、レジスタ番号2については有効範囲が重複しているため、レジスタ番号0x1cをレジスタ番号0に置き換えることとする。これにより、図19(b)に示したようなプログラムを得ることができる。

【0089】プログラムの置き換えが終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する(ステップS1805)。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップS1803～S1805を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【0090】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてCPU等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【0091】なお、本実施例では、実行ユニットおよび命令バスがともに32ビットの場合を例に採って説明したが、複数ワードのアドレスを同時に転送するような場合にも、第2の発明を適用できることはもちろんである。例えば4ワードバウンダリで転送する場合にも、注目する命令の4ワード前の命令および4ワード後の命令とのハミング距離について評価を行えばよい。

【0092】(実施例3)次に、実施例3として、第3の発明の一実施例(請求項7に対応する)について説明する。

【0093】本実施例では、“add”命令に第3の発明を適用した場合を例に採って説明する。

【0094】命令機能コードを作成するとき、例えばSPARCの命令機能コードを参照すると、“add”命令は“000000”であるが、この“add”命令等は非常に出現頻度が高い命令であるので、本実施例では、“000000”に加えて“111111”も“add”命令となるように、命令体系を作成する。すなわち、本実施例の命令列最適化装置で最適化された制御プログラムを使用する情報処理装置においては、“000000”および“111111”を“add”命令であるとしてデコードするように、CPUの命令デコーダが構成されるものとする。これにより、コンパイラは、オブジェクト・コードを作製する際に、命令機能コードのフィールドに“000000”または“111111”のいずれかを割り当てることができる。ここで、“add”命令の前後の命令の命令機能コードのフィールドが“001110”および“110110”であったとすると、“add”命令の命令機能コードに“000000”を選択した場合のハミング距離は7であり、“add”命令の命令機能コードに“111111”を選択した場合のハミング距離は5である。したがって、この場合には、コンパイラは、“add”命令の命令機能コードに“111111”を選択する。

【0095】次に、本実施例の命令列最適化装置の具体的な例について、図21および図22を用いて説明する。

【0096】図21は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【0097】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード(アセンブリ・コード)を作製する(ステップS2101)。このとき、“add”命令の命令機能コードは“000000”となっているものとする。

【0098】次に、各命令について、本実施例を適用する命令であるか否か、すなわち命令機能コードのフィールドを複数割り当てられている命令(ここでは“add”命令)であるか否かを、判断する(ステップS2102)。

【0099】そして、本実施例を適用する命令であると判断された場合は、この命令に対して、置換が可能なビットパターン(ここでは“111111”)を選出する(ステップS2103)。

【0100】さらに、この命令に対応するビットパターンのすべてについて、その前後の命令とのハミング距離を算出し、互いに比較することによって、ハミング距離が最低となるようなビットパターンを選択する(ステップS2104)。図22に、“add”命令についての最適化処理を行ったプログラムの例を示す。この例では、先に現れた“add”命令では命令機能コードを“000000”とした方がハミング距離が小さいので置換を行わず、後に現れた“add”命令では命令機能コードを“111111”とした方がハミング距離が小さいので置換を行っている。

【0101】プログラムの置換が終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する(ステップS2105)。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップS2103～S2105を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【0102】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてCPU等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【0103】(実施例4)次に、実施例4として、第4の発明の一実施例(請求項8に対応する)について説明する。

【0104】本実施例では、1種類の動作を行うための実現方法が複数ある場合に、前後の命令とのハミング距離が最も小さくなるように、その実現方法に係わる命令を選択する。例えば、レジスタ0x0dにデータ“0”を書き込む場合、SPARCのようにレジスタ番号0が書き込みは意味がない読み出しはデータ“0”を出力する特別なレジスタとして定義されている場合、その実現方法としては、以下のようなものがある。これらの実現方法(すなわち命令)のうちで、前後の命令とのハミング距離が最も小さくなるものを選択して、その命令を置き換えることとする。

【0105】

mov r0, rd(0x0dにデータ“0”を移す命令)

add r0, r0, rd(0+0を0x0dに格納させる命令)

mul r?, r0, rd(ある値に0を掛けた値を0x0dに格納させる命令)

mul r0, r?, rd(0にある値を掛けた値を0x0dに格納させる命令)

xor r?, r?, rd(ある値と、これと同じ値との排他的論理和を取った結果を0x0dに格納させる命令)

sll r0, r?, rd(0をある値だけ右にシフトさせた値を0x0dに格納させる命令)

srl r0, r?, rd(0をある値だけ左にシフトさせた値を0x0dに格納させる命令)

また、他の具体例としては、イミディエイト加算命令によるものがある。イミディエイト加算に対して、イミディエイト部分を2の補数としたイミディエイト減算は、演算機能としてはまったく同じである。例えば、 $a=b+5$ と、 $a=b-(−5)$

とは、同じ演算として扱われる。ここで、イミディエイト加算とイミディエイト減算とを置き換えた場合、イミディエイトデータを表す命令のフィールドが反転するので、両式を置き換えることによってハミング距離を低減することができる場合がある。

【0106】本実施例において、置き換えを行う候補を選出するためには、例えば、いわゆるライブラリを予め作製しておき、ある命令を評価するときにこのライブラリに置き換え候補が登録されているか否かを検索すればよい。検索の結果、置き換え候補が検索された場合には、この置き換え候補を採用した場合とハミング距離を比較する。また、イミディエイト加算とイミディエイト減算とを置き換えるためには、ライブラリに、イミディエイト加算の置き換え候補として、イミディエイト減算を登録しておけばよい。このとき、イミディエイトデータを変換する方式或いは手順もライブラリに登録しておけば、イミディエイト減算が検索されることによって変換方式・手順も得られるようにすることができる。例えば、命令フィールドをイミディエイト減算に置き換えるとともに、イミディエイトデータを2の補数を取ったものに置き換えるといった手順を採用することができる。そして、このような手順で得られた命令データを、ハミング距離の比較対象として採用する。また、単にライブラリの検索を行うのではなく、検索が可能であるか否かを判断した後で、可能である場合には検索を行うこととしてもよい。

【0107】次に、本実施例の命令列最適化装置の具体的な例について、図23および図24を用いて説明する。

【0108】図23は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【0109】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード(アセンブリ・コード)を作製する(ステップS2301)。

【0110】次に、各命令について、本実施例を適用する命令であるか否か、すなわちライブラリに置き換え候補が登録されている命令であるか否かを、判断する(ステップS2302)。

【0111】そして、本実施例を適用する命令であると判断された場合は、ライブラリの検索を行って、置換が可能な命令を選出する(ステップS2303)。また、このとき、命令動作を解析して同等の命令を生成することとしてもよい。

【0112】さらに、本実施例を適用する命令およびライブラリで検索された命令について、その前後の命令とのハミング距離を算出する。そして、各算出結果を互いに比較することによって、ハミング距離が最低となるような命令を選択する(ステップS2304)。図24において、(a)は本実施例による最適化を行う前のプログラム例であり、(b)は最適化後のプログラム例である。同図において“add”命令(イミディエイト加算命令)を“subi”命令(イミディエイト減算命令)に置き換えることにより、その前後の命令との間のハミング距離を26から20に低減させることができた。

【0113】命令の置換が終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する(ステップS2305)。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップS2303～S2305を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【0114】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてCPU等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【0115】(実施例5)次に、実施例5として、第5の発明の一実施例(請求項9に対応する)について説明する。

【0116】本実施例では、1種類の動作を行うための実現方法が複数ある場合に、作動する機能ブロックが小さく、消費電力が小さくなるように、その実現方法に係わる命令を置き換える。すなわち、データ線のばらつき、使用する機能ブロックの消費電力などを考慮し、総合的な消費電力が最小となるように、命令の置き換えを行う。置き換えの方法としては、上述の実施例4の場合と同様、ライブラリを使用することができる。例えば、レジスタ0x0dにデータ“0”を書き込む場合、採用する命令と使用する機能ブロックとの関係は、表1のようになる。これらの命令のうちで、消費電力が最も小さくなるものを選択して、その命令を置き換えることとする。

【0117】

【表1】

命 令	使用する機能ブロック
mov r0, rd	バイパス回路
add r0, r0, rd	加算器
mul r?, r0, rd	乗算器
mul r0, r?, rd	乗算器
xor r?, r?, rd	ロジック回路
sl1 r0, r?, rd	バレルシフト
sr1 r0, r?, rd	バレルシフト

次に、本実施例の命令列最適化装置の具体的な例について、図25を用いて説明する。

【0118】図25は、本実施例の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【0119】まず、高級言語或いはアセンブリ言語で作製されたソース・プログラムをコンパイルし、さらに他の最適化処理を施すことにより、中間コード(アセンブリ・コード)を作製する(ステップS2501)。

【0120】次に、各命令について、本実施例を適用する命令であるか否か、すなわちライブラリに置き換え候補が登録されている命令であるか否かを、判断する(ステップS2502)。

【0121】そして、本実施例を適用する命令であると判断された場合は、ライブラリの検索を行って、置換が可能な命令を選出する(ステップS2503)。また、このとき、命令動作を解析して同等の命令を生成することとしてもよい。

【0122】さらに、本実施例を適用する命令およびライブラリで検索された命令について、消費電力を試算する。そして、各算出結果を互いに比較することによって、消費電力が最低となるような命令を選択する(ステップS2504)。

【0123】命令の置換が終了すると、続いて、すべてのデータについて最適化が終了したか否かを判定する(ステップS2505)。そして、最適化が終了していないデータが残っている場合には、そのデータについてステップS2503～S2505を実行する。一方、すべてのデータについて最適化が終了している場合には、最適化処理後のプログラムを出力し、最適化処理を終了する。

【0124】本実施例によれば、以上説明したようにして最適化を行った制御プログラムを情報処理装置のプログラムメモリに格納し、この制御プログラムを用いてCPU等の制御を行うことにより、命令バスにおける消費電力を低減させることが可能となる。

【0125】

【発明の効果】以上詳細に説明したように、本発明によれば、情報処理装置用制御プログラムの作成段階で消費電力を低減させるための最適化処理を行うことができる、命令列最適化装置を提供することができる。

図の説明

【図面の簡単な説明】

【図1】実施例1の概念を概略的に示すフローチャートである。

【図2】図1を具体化した例を示すフローチャートである。

【図3】実施例1に係わる命令列最適化装置で最適化されるプログラムの一例を示す図である。

【図4】図3に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図5】図3に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図6】図3に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図7】図3に示したプログラムをコンパイルしたアセンブリ・ソース・プログラムリストを示す図である。

【図8】図3における基本ブロックの依存関係の解析結果を示す有向グラフである。

【図9】図3における基本ブロックの依存関係の解析結果を示す有向グラフである。

【図10】図3における基本ブロックのビット・パターンを示す図である。

【図11】実施例1による最適化処理後のビット・パターンを示す図である。

【図12】実施例1による最適化処理の効果の説明するためのビット・パターンを示す参考図である。

【図13】実施例1の変形例を説明するためのフローチャートである。

【図14】(a)、(b)ともに、実施例1で最適化されたプログラムを使用する装置の一構成例を示す概念図である。

【図15】(a)、(b)ともに図14に示した装置で使用するプログラムのビット・パターンを示す図である。

【図16】(a)は実施例1で最適化されたプログラムを使用する装置の一構成例を示す概念図、(b)は(a)に示した装置で使用するプログラムのビット・パターンを示す図である。

【図17】実施例1で最適化されたプログラムを使用する装置の一構成例を示す概念図である。

【図18】実施例2の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図19】(a)は実施例2の命令列最適化装置が行う最適化処理で使用する中間コードのプログラム例を示す図、(b)は(a)のプログラムを最適化した結果を示す図である。

【図20】図19(a)に示したプログラムの有効範囲テーブルを示す図である。

【図21】実施例3の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図22】実施例3の最適化処理を“add”命令について行ったプログラムを示す図である。

【図23】実施例4の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図24】(a)は実施例4による最適化を行う前のプログラムの一例を示す図、(b)は(a)のプログラムを最適化した後のプログラムを示す図である。

【図25】実施例5の命令列最適化装置が行う最適化処理の手順を説明するためのフローチャートである。

【図26】携帯型情報処理装置の制御部の概略構成を示すブロック図である。

【符号の説明】

2610 CPU

2611 実行ユニット

2612 入出力部

2613 レジスタ部

2620 プログラムメモリ

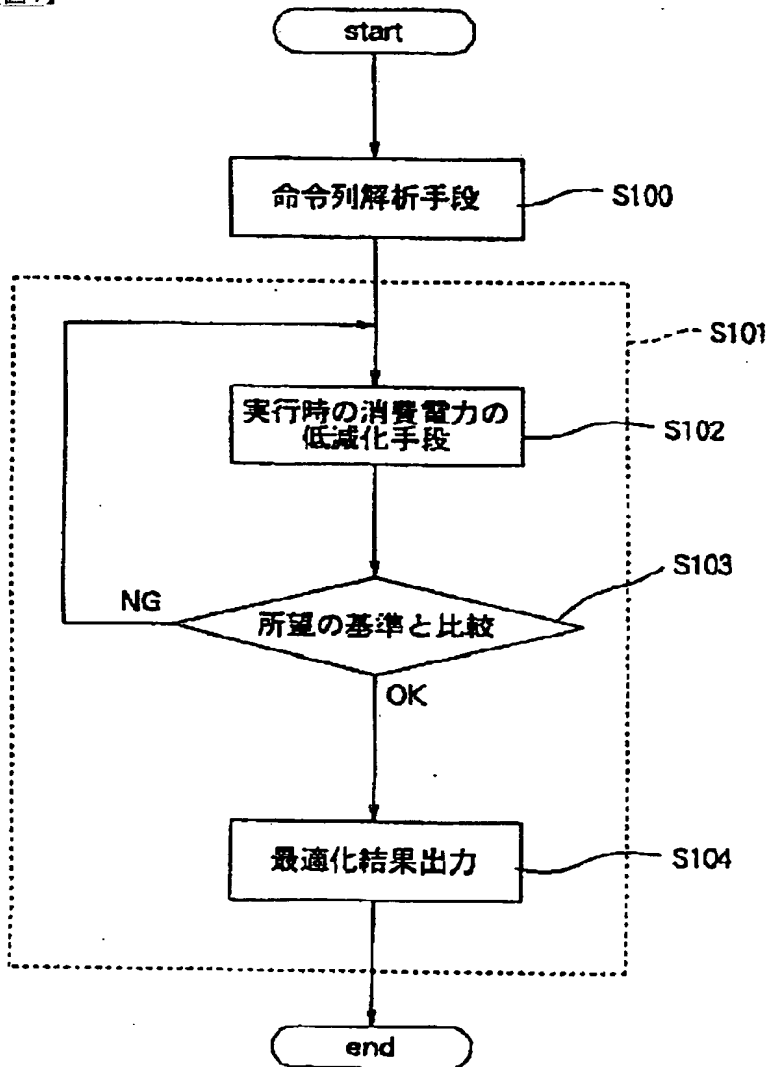
2621 記憶部

2631 アドレスバス

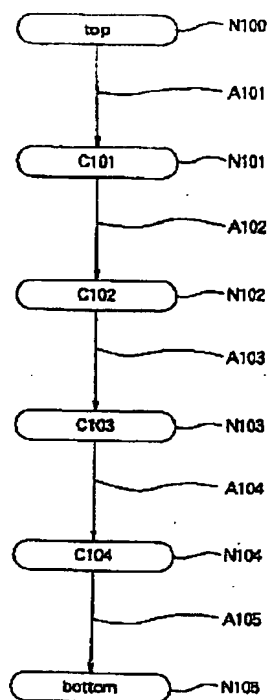
2632 命令バス

図面

【図1】



【図8】



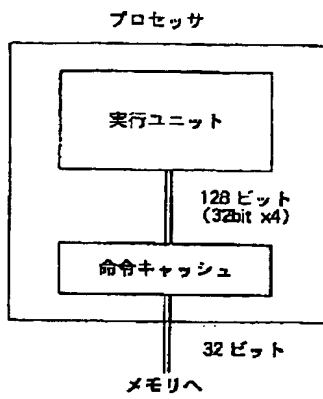
【図10】

C204: 000000010000000000000000000000
 C301: 10010000001001111010000110011000
 C302: 1101001000000111101111111111100
 C303: 1001010100101010011000000000010
 C304: 110100000000010000000000001010
 C305: 10010110001001111010001100101000
 C306: 1101100000000111101111111111100
 C307: 1001101100101011001000000000010
 C308: 1101001000000010110000000001101
 C309: 01000000000000000000000000000000
 C310: 00000001000000000000000000000000
 C311: 11011110000001111011111111111000
 C312: 101000000000011110000000001000
 C313: 11100000001001111011111111111000

【図11】

C204: 00000001000000000000000000000000
 C301: 10010000001001111010000110011000
 C306: 1101100000000111101111111111100
 C302: 1101001000000111101111111111100
 C305: 10010110001001111010001100101000
 C307: 1001101100101011001000000000010
 C303: 1001010100101010011000000000010
 C304: 110100000000010000000000001010
 C308: 1101001000000010110000000001101
 C309: 01000000000000000000000000000000
 C310: 00000001000000000000000000000000
 C311: 11011110000001111011111111111000
 C312: 101000000000011110000000001000
 C313: 11100000001001111011111111111000

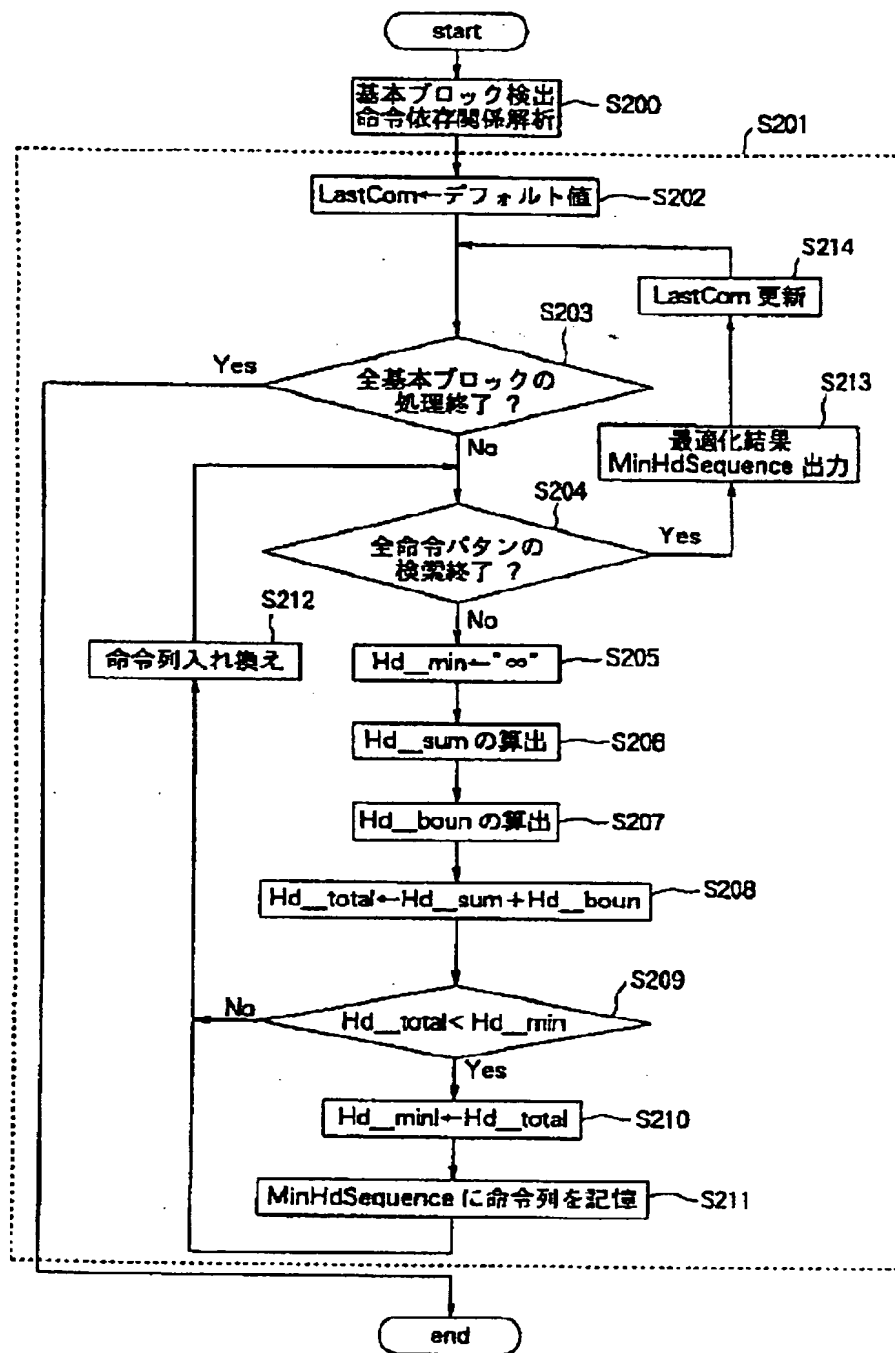
【図17】



【図20】

レジスタ番号	開始アドレス	終了アドレス
0x01	0011	0101
0x02	0100	0110
0x03	0110	0111
0x04	0000	0110
0x05	0001	1011
0x08	0010	1100
0x1c	0101	1000
0x01	1011	1101
0x02	1100	1101
0x1c	1101	1110

【図2】



【図22】

```

div r13,r16,r11
add r17,r5,r17
div r17,r14,r14
add r11,r5,r17
mul r17,r14,r14

```

```

010111 10001 10011 000 11110101 10110
111111 10111 10001 010 00000000 00101
010111 10101 10011 000 11110101 10111
000000 10111 10001 010 00000000 00101
010010 10101 10011 000 11110101 10111

```

【図3】

```

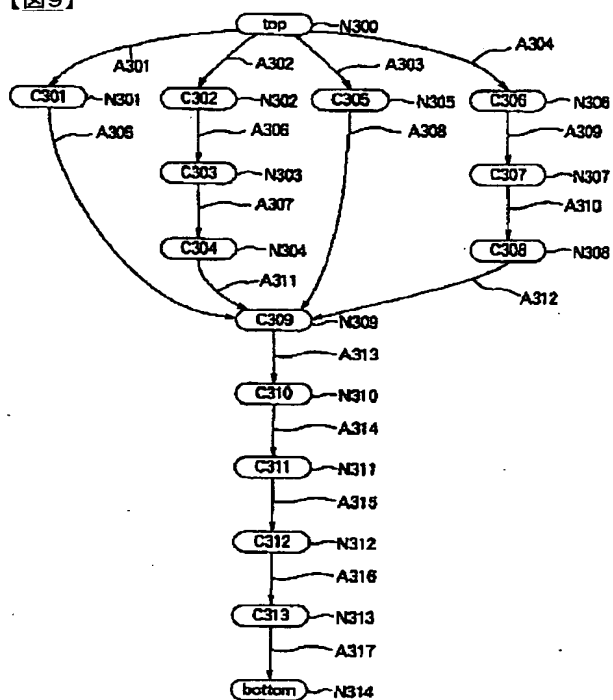
/* Sample Program */

#include <stdio.h>
#define MAX 100

main()
{
    int i, q, z[MAX], x[max];
    for (i=0; i < MAX; i++) q = q + z[i] * x[i];
    printf("Q = Q + Z[i] x X[i] : %d\n", q);
}

```

【図9】



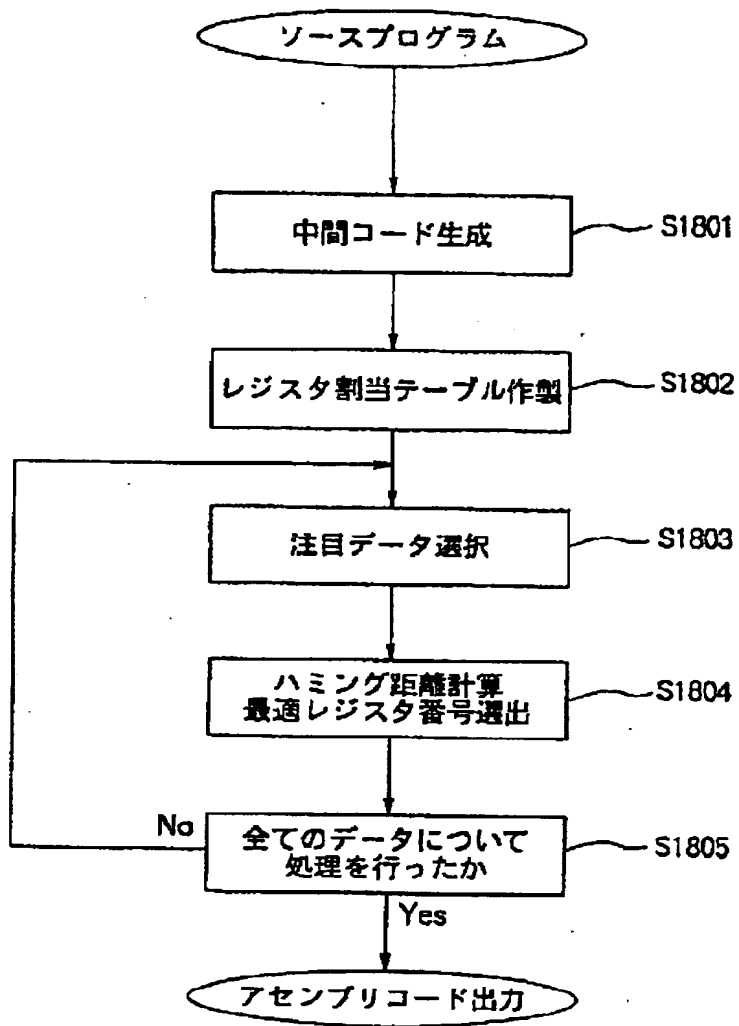
【図12】

```

C204: 00000001000000000000000000000000
C302: 11010010000001111011111111111100
C303: 1001010100101010011000000000010
C306: 1101100000000111011111111111100
C307: 1001101100101011001000000000010
C301: 10010000001001111010000110011000
C308: 1101001000000010110000000001101
C305: 10010110001001111010001100101000
C304: 110100000000010000000000001010
C309: 01000000000000000000000000000000
C310: 00000001000000000000000000000000
C311: 11011110000001110111111111111000
C312: 101000000000011100000000001000
C313: 11100000001001111011111111111000

```

【図18】



【図4】

```

*****
LineNo  Address      Object Code      Assembly Source
*****
1      1      LL0:
2      2      .seg      "data"
3      3      .seg      "text"
4      4      .proc 04
5      5      .global __main
6      6
7      7
8      8      00000000      033fffff      ! # PROLOGUE # 0      sethi      %hi(LF25),%g1      ..... C101
9      9      00000004      82006078      add      %g1,%lo(LF25),%g1      ..... C102
10     10     00000008      9de38001      save     %sp,%g1,%sp      ..... C103
11     11
12     12     0000000c      c027bffc      ! # PROLOGUE # 1      st      %g0,[%fp+-0x4]      ..... C104
13     13
14     14     00000010      d007bffc      L29:      ld      [%fp+-0x4],%a0      ..... C201
15     15     00000014      80a22064      cmp      %o0,0x64      ..... C202
16     16     00000018      16800014      dge      L28      ..... C203
17     17     0000001c      01000000      nop      ..... C204

```

【図5】

18	00000020	9027a198	sub	%fp, 0x198, %o0	C301	B3
19	00000024	d207bffc	ld	[%fp + -0x4], %o1	C302	
20	00000028	952a6002	sil	%o1, 0x2, %o2	C303	
21	0000002c	d002000a	ld	[%o0 + %o2], %o0	C304	
22	00000030	9627a328	sub	%fp, 0x328, %o3	C305	
23	00000034	d807bffc	ld	[%fp + -0x4], %o4	C306	
24	00000038	9b2b2002	sil	%o4, 0x2, %o5	C307	
25	0000003c	d202c00d	ld	[%o3 + %o5], %o1	C308	
26	00000040	40000000	call	.mul, 2	C309	
27	00000044	01000000	nop		C310	
28	00000048	de07bffb	ld	[%fp + -0x8], %o7	C311	
29	0000004c	a003c008	add	%o7, %o0, %l0	C312	
30	00000050	e027bffb	st	%l0, [%fp + -0x8]	C313	
31						B4
32	00000054	e207bffc	ld	[%fp + -0x4], %l1		

L27 :

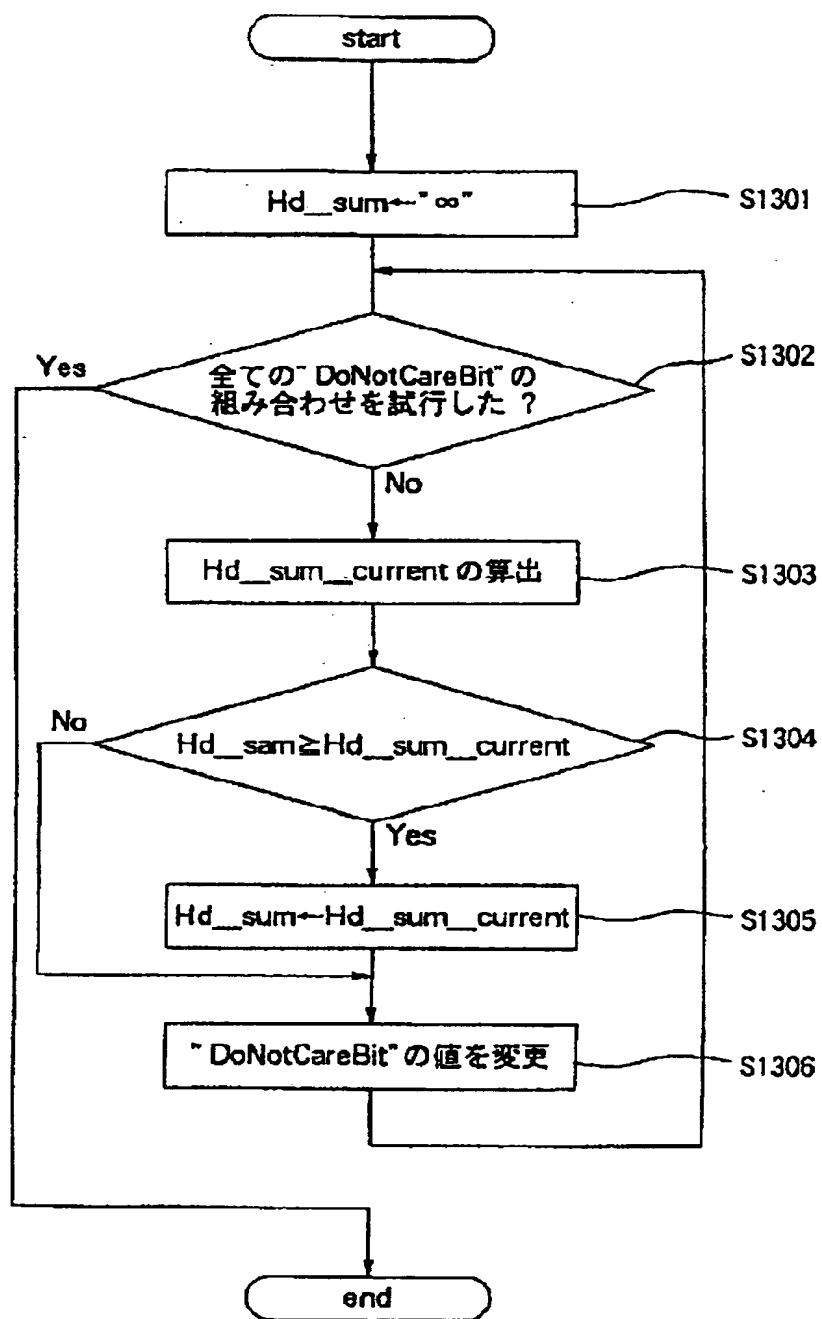
【図6】

33	00000058	a2046001	add	%l1, 0x1, %l1	B4
34	0000005c	e227bffe	st	%l1, [%fp+0x4]	
35	00000060	10b1ffec	b	L29	
36	00000064	01000000	nop		
37					
38			.seg	"data1"	
39					
40	%d*012*0"		.ascii	*Q = Q + Z[i] * X[i] :	
41					
42	00000068	11000000	.seg	"text"	B5
42	0000006c	90122000	set	L31, %c0	
43	00000070	d207b1f8	ld	[%fp+0x8], %o1	
44	00000074	40000000	call	__printf, 2	
45	00000078	01000000	nop		
46	0000007c	90102000	mov	0, %c0	

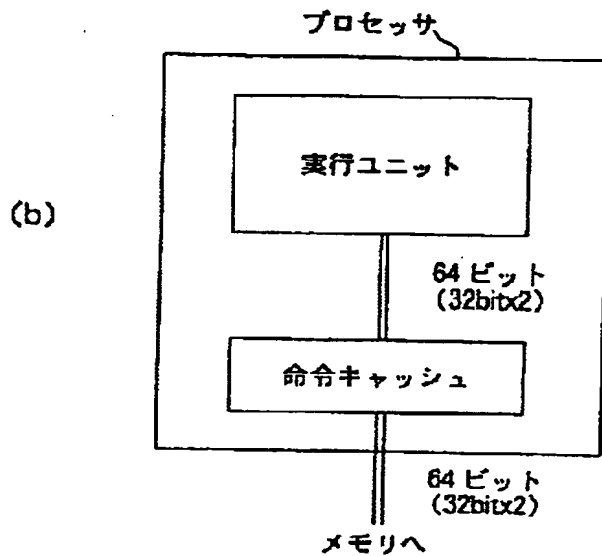
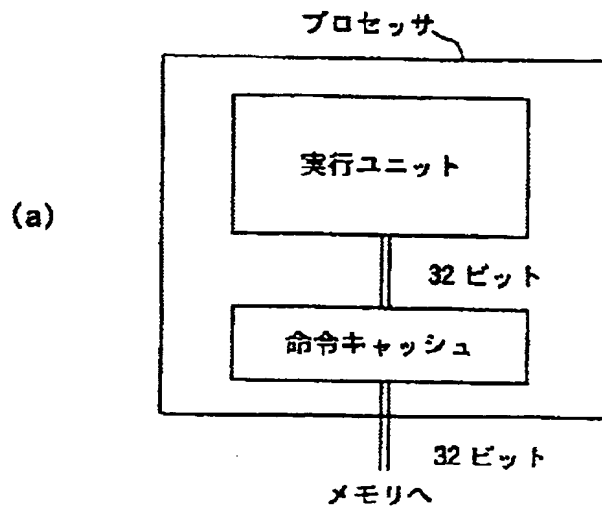
L28 :

L31 :

【図7】



【図14】



【図24】

(a)

mul r13,r16,r11	010010 10001 10011 000 11110101 10110
addi r11,5,r17	010000 10111 10001 010 000000000101
mul r17,r14,r14	010010 10101 10011 000 11110101 10111

(b)

mul r13,r16,r11	010010 10001 10011 000 11110101 10110
subi r11,5,r17	010001 10111 10001 010 1111111111011
mul r17,r14,r14	010010 10101 10011 000 11110101 10111

【図15】

(a)

C301 : 10010000001001111010000110011000
 C302 : 1101001000000111011111111111100
 C303 : 1001010100101010011000000000010
 C304 : 1101000000000100000000000001010
 C305 : 10010110001001111010001100101000

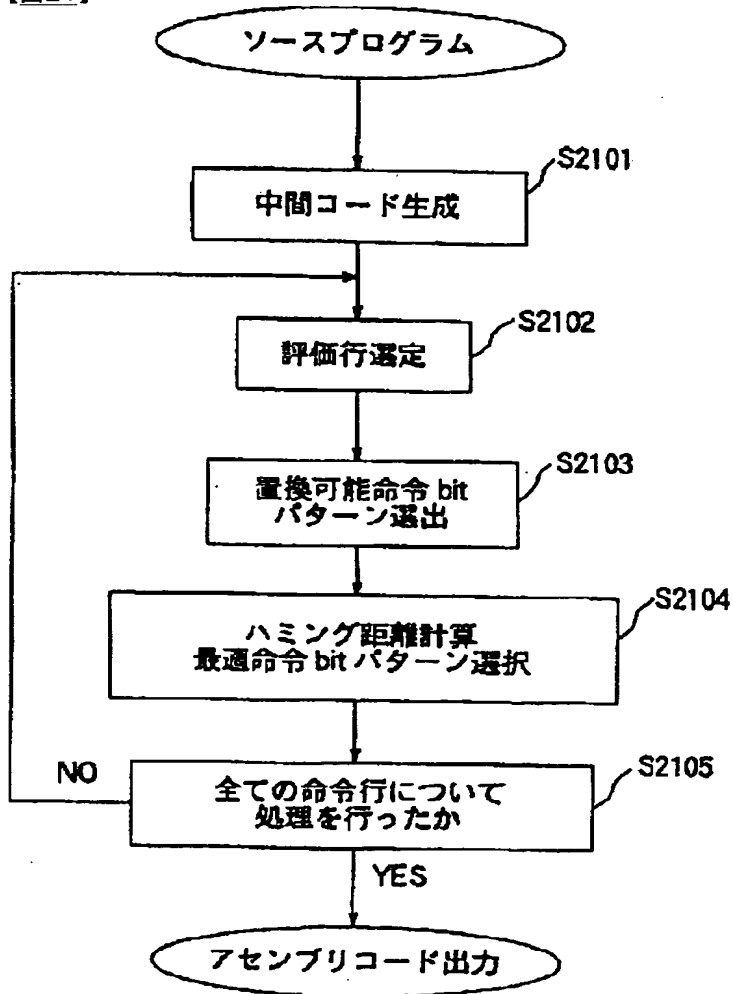
 C310 : 00000001000000000000000000000000
 C311 : 11011110000001110111111111111000
 C312 : 1010000000000011100000000001000
 C313 : 11100000001001111011111111111000

(b)

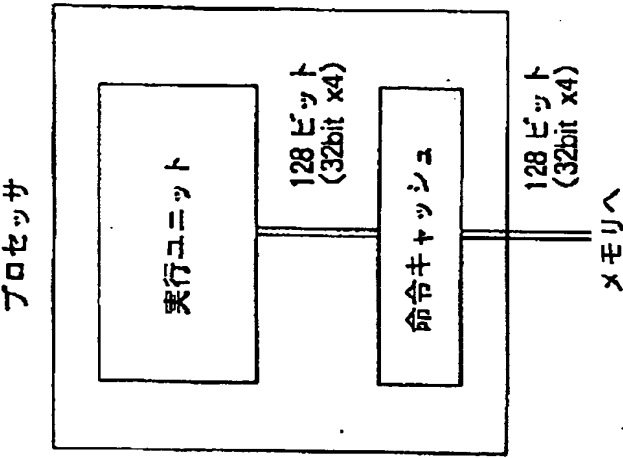
C301, C302 : 10010000001001111010000110011000 1101001000000111011111111111100
 C303, C304 : 1001010100101010011000000000010 1101000000000100000000000001010

 C310, C311 : 00000001000000000000000000000000 11011110000001110111111111111000
 C312, C313 : 1010000000000011100000000001000 11100000001001111011111111111000

【図21】



【図16】



(a)

C301, C302, C303, C304: 10010..11000 11010..11100 10010.00010 11010.01010
.....
C309, C310, C311, C312: 01000..00000 00000..00000 11011..11000 10100..01000

(b)

【図19】

(a)

LABEL :

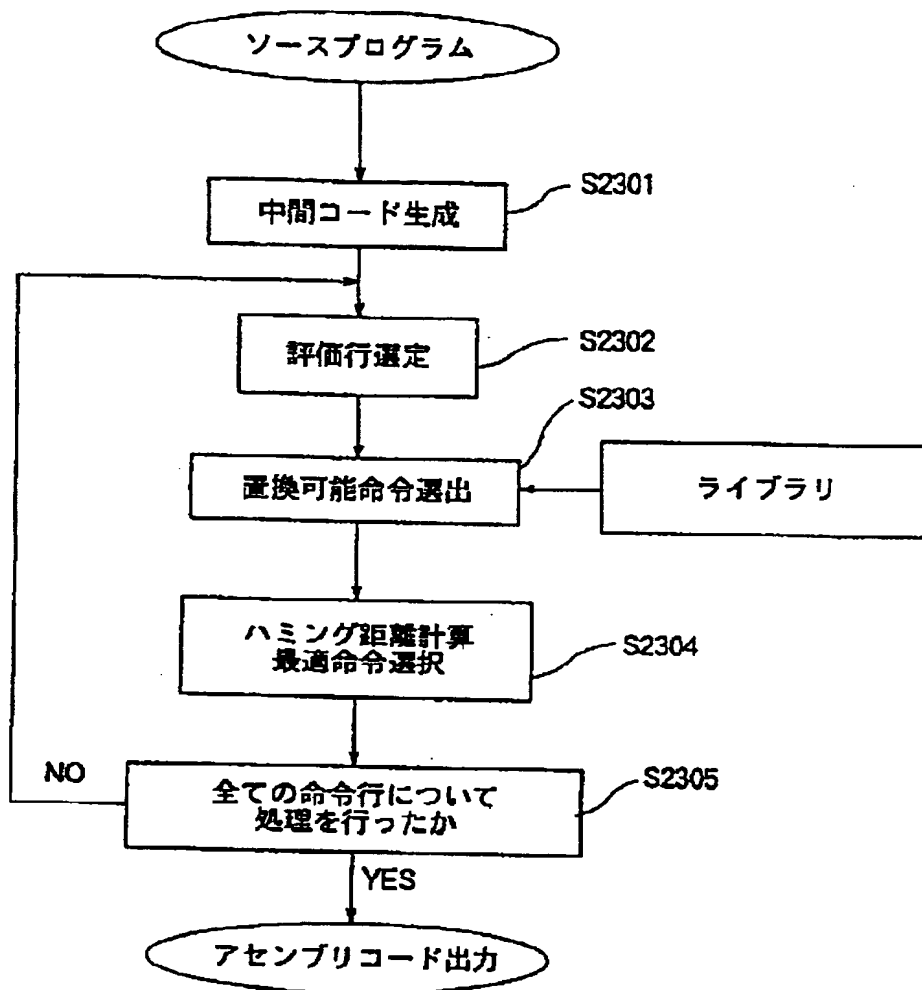
0000	set 0x00101 r4	1011010000 00100 0000000001000 00001
0001	set 0x00215 r5	1011010000 00101 0000000010000 10101
0010	set 0x00112 r6	1011010000 00101 0000000001000 10010
0011	ld [r5+0x4] r1	1100000001 00001 00101 0000000 00100
0100	ld [r6+0x4] r2	1100000001 00010 00110 0000000 00100
0101	mul r1 r2 r1c	1100101011 <u>11100</u> 00001 0000000 00010
0110	add r2 r4 r3	1100101011 00011 00010 0000000 00100
0111	sub r1c r3 r1c	1100011011 <u>11100</u> 00011 0000000 <u>11100</u>
1000	cmp r1c 0x4c	1011010000 <u>11100</u> 00000 0000010 01100
1001	bne LABEL	0100000010 00000 11111 1111111 11100
1010	nop	0000000000 00000 00000 0000000 00000
1011	ld [r5+0x20] r1	1100000001 00001 00101 0000001 00000
1100	ld [r6+0x20] r2	1100000001 00010 00110 0000001 00000
1101	mul r1 r2 r1c	1100101011 <u>11100</u> 00001 0000000 00010
1110	st [r6+0x28] r1c	1100100111 <u>11100</u> 00100 0000001 01000

(b)

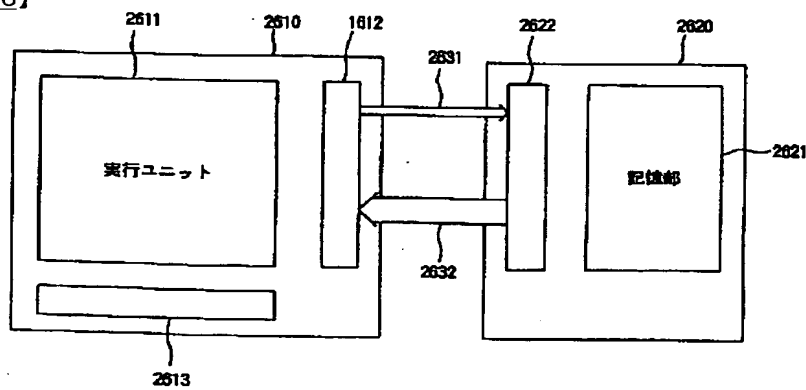
LABEL :

0000	set 0x00101 r4	1011010000 00100 0000000001000 00001
0001	set 0x00215 r5	1011010000 00101 0000000010000 10101
0010	set 0x00112 r6	1011010000 00101 0000000001000 10010
0011	ld [r5+0x4] r1	1100000001 00001 00101 0000000 00100
0100	ld [r6+0x4] r2	1100000001 00010 00110 0000000 00100
0101	mul r1 r2 r0	1100101011 <u>00000</u> 00001 0000000 00010
0110	add r2 r4 r3	1100101011 00011 00010 0000000 00100
0111	sub r0 r3 r0	1100011011 <u>00000</u> 00011 0000000 <u>00000</u>
1000	cmp r0 0x4c	1011010000 <u>00000</u> 00000 0000010 01100
1001	bne LABEL	0100000010 00000 11111 1111111 11100
1010	nop	0000000000 00000 00000 0000000 00000
1011	ld [r5+0x20] r1	1100000001 00001 00101 0000001 00000
1100	ld [r6+0x20] r2	1100000001 00010 00110 0000001 00000
1101	mul r1 r2 r1c	1100101011 <u>11100</u> 00001 0000000 00010
	(有効範囲外なので変更せず)	
1110	st [r6+0x28] r1c	1100100111 <u>11100</u> 00100 0000001 01000

【図23】



【図26】



【図25】

